Low Energy Digit-serial Architectures for large $GF(2^m)$ multiplication

FETH ALLAH CHERIGUI, DANIEL MLYNEK

Integrated Systems Laboratory

Swiss Federal Institute of Technology EPFL

CH-1015 Lausanne

Feth-Allah.Cherigui@epfl.ch
+41 21 693 6983
Daniel.Mlynek@epfl.ch
+41 21 693 4681

Abstract. This paper presents two low-energy, highly regular, VLSI architectures performing a large prime $GF(2^m)$ multiplication. The first one is area-efficient digit-serial architecture, when field-generating polynomial $p(x)$ is a trinomial. The second architecture is digit-serial, programmable on $p(x)$. Both architectures are suitable for computing large prime $GF(2^m)$ exponentiation for DL based schemes. The parallel algorithm inside of each digit cell reduces both the global cycle time for the first architecture and the switching activity in the second one. An analysis of the performance comparison is described as function of the digit-size. A comparison is made with the bit serial architecture based on the performance improvement with respect to computation delay and energy consumption of one multiplication operation. Thus, the factor of merit for performance measurement is defined as the product of *energy times the delay* and it is computed. The simulation results on gate level implementations shows that the energy delay products are highly reduced for both architectures. Therefore, the proposed architectures are attractive for low-power applications.

Key words. Finite Field multiplier, Trinomials, Digit-serial architecture, Low-power design, Low-energy design, Energy-Delay product, Switching Activity, Bit-level Pipelining, Long Heavily Loaded Lines.

1. INTRODUCTION

Finite field arithmetic architectures are the basic building blocks in many applications involving cryptography. Many popular public-key algorithms require exponentiation in large Galois Field $GF(2^m)$, including in particular schemes based on the intractable discrete logarithm in finite fields [1].

This operation can be computed by repeated square-and-multiply (S & M) algorithm [7] as a series of modular multiplications over $GF(2^m)$. Although, hardware integration of large prime field $GF(2^m)$ multipliers present a high degree of complexity related to the field-size and field-generating polynomial. Furthermore, the long arithmetic operators exhibit in general a great activity and dissipate consequent shares of the power supply.

Reducing power consumption is equally important for non-portable systems as it reduces cooling and packaging costs and increases system reliability. Thus, the design of efficient dedicated, low energy, finite field multipliers can lead to dramatic improvement on the overall system performance of $GF(2^m)$ exponentiator.

The usual approach to reduce the time complexity and improve the performance is to use parallel multipliers. However, the hardware complexity of a bit-parallel multiplier is proportional to m^2. Its area and energy consumption increase dramatically as the field order m increase since large number of gates and registers are mapped.

Digit-serial technique an alternative to the bit-parallel approach, process multiple bits "digit" of an entire word, referred to as the *digit-size*, in one clock-cycle. This technique is suitable for the implementation of moderate sample rate systems where, the area and power consumption are critical. It was first used for the implementation of Galois Field multiplier in [15]. However, the architecture of the multiplier is based on semi-systolic 2-D array multiplier architecture [10] in which, a large amount of gates and registers have to be mapped yielding to increase both the area and the power consumption for large field-size. In this paper a new digit-serial, high performance $GF(2^m)$ multipliers are developed and implemented. The proposed architectures are mapped on low-power/low-voltage technology. A technique such as gating the clock is used to analyze the amount of power savings using different digit-size.

The outline of the paper is as follows. Section 2 provides Knowledge of basic Finite Field concepts and properties, then some considerations are discussed concerning the Primitive polynomials and field-size for DL based Finite Field

based cryptosystems and a brief overview of the existing VLSI architectures for performing multiplication in $GF(2^m)$. Two selected architectures are briefly exposed. In section 3, the corresponding theoretical basis for our proposed digit-serial multiplication algorithms are developed and special purpose architectures for implementing the proposed algorithms are described. The implementation results and comparison are detailed in Section 4. and some conclusions are provided in Section 5.

2. Finite Field $GF(2^m)$ Survey

2.1. Mathematical Background

Knowledge of basic Finite Field concepts and properties is assumed, as covered in [4], [16].

Finite Field $GF(2^m)$ contains 2^m elements. It is an extension field of $GF(2)$, which contains two elements {0,1}. The element of $GF(2^m)$ can be represented in several equivalent forms. Mainly, there are three common types of bases, *Standard or Polynomial Basis* (SD/PB), *Normal Basis* (NB) and *Dual basis* (DB). There are many polynomial bases and normal bases from which to choose. For efficient computation of the field arithmetic we generally use an *optimal normal basis representation* or a *polynomial basis representation*.

If a standard basis $\{1, \alpha^1, \ldots, \alpha^{m-1}\}$ is used, where the primitive element α is a root of an irreducible polynomial of degree m, $p(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1 x + p_0$ over $GF(2)$, then

$$\alpha^m = \sum_{i=0}^{m-1} p_i \alpha^i \qquad (1)$$

Each element can be represented as a polynomial in α with a degree less then m, or

$$GF(2^m) = \left\{ A \middle| A = \sum_{i=0}^{m-1} a_i \alpha^i,\ a_i \in GF(2),\ 0 \le i \le m-1 \right\} \qquad (2)$$

In addition, the operation results of additions, multiplications and exponentiation of element a are still polynomials of a with degree less than *m*. In this base, addition is defined as integer addition modulo-2 (logical XOR) and multiplication is defined as integer multiplication modulo-2 (logical AND). Element of the field represented by a normal basis $\left\{a, a^2, a^4, ..., a^{2^{m-1}}\right\}$, are expressed as polynomials of degree 2^{m-1} or less, or

$$GF(2^m) = \left\{ A \middle| A = \sum_{i=0}^{m-1} a_i a^{2^i},\ a_i \in\ GF(2),\ 0 \le i \le m-1 \right\} \qquad (3)$$

Since elements in one representation can be efficiently converted to elements in the other representation by using an appropriate change-of-basis matrix, the intractability of the DLP isn't affected by the choice of representation.

2.2. Primitive Polynomials and Field Size for Finite Field based Cryptosystems

When first introduced as underlying Finite Field, $GF(2^m)$ was the preferred implementation, basically because it is easier to implement in hardware [2], [3] using LFSRs. Although, all practical DL based public-key schemes require operations in relatively large Finite Fields; e.g., $m > 500$ bits [3][19]. Further, for security reasons, the field-size *m* is selected so that 2^m-1 is a large prime (a "Mersenne" prime). There are certain values of field-size for which the period of the LFSR is the maximum, namely 2^m-1, which is all the possible states of *m* bits, excluding the all-zero state. A maximum length sequence will occur if the reduction polynomials for constructing extension field corresponding to the LFSRs are prime elements of $GF(2^m)$ [4].

In addition, arithmetic in $GF(2^m)$ can usually be implemented more efficiently if the chosen irreducible polynomial has few non-zero terms. Since, the least significant coefficient of any prime polynomial must always be nonzero (otherwise the polynomial has 0 as a root), the *hamming* weight of the prime polynomials of degree at least 2 with few nonzero coefficients, must be odd and have at least 3 coefficients (prime polynomials of *low hamming weight*). Polynomials of hamming weight 2, 3, 4 are called *trinomials*, *quadrinomials* and

pentanomials respectively. An irreducible trinomial of degree m must be of the form x^m+x^k+1, where $1 \le k \le m-1$. In fact, both the complexity and energy consumption of $mod\ p(x)$ operation could be significantly reduced by selecting k with smaller value and less Hamming weight [5]. Table 1., gives some practical values for parameter k and, the field-size m, for which an irreducible trinomial of degree m in Finite Field exists.

Table 1 Most useful irreducible trinomials x^m+x^k+1, for each large Mersenne prime m, $512 \le m \le 4423$

Table 1

2.3. Architectures for $GF(2^m)$ Multiplication

Various architectures have been proposed to perform modular multiplication operation efficiency in $GF(2^m)$. Different basis representation, have been used to obtain some interesting realizations [4][8][9][10][11]. The parallel approaches, aren't to be enumerate here, since for large m the multiplier has to be of serial type. In fact for an arbitrary $GF(2^m)$ the gate count for a bit-parallel multiplier using either a PD or NB is proportional to m^2. In that case, area complexity increases dramatically for large field size.

A synthesis comparison among DB, NB and SB is given in [12][13]. There, the gate count is a guideline for the implementation complexity. It shows that, multipliers based on NB and DB requires basis conversion. Moreover, the area of NB multiplier grows dramatically as the order of the field goes up when optimal normal basis doesn't exist. Even when an optimal normal basis is chosen, the size complexity is proportional to $3m$. Also, both DB and NB are not highly modular or expandable [14]. Instead, the SD multiplier does not require basis conversion, its size and time complexity are proportional to m and it is readily matched to any input or output system [12]. The polynomial basis multiplier can be implemented using different architectures. The systolic and/or semi-systolic multipliers are described in [9][10]. These architectures are pipelined and regular 2-D systolic arrays based on approach similar to the bit-serial one (MSR) [4]. Their hardware implementations use m bit-serial parallel multipliers resulting in expensive

hardware offering a high bit rate. The systolic multiplier described in [9] and [10] is thus not very attractive for large Finite Field.

The MSR architecture described in [4] is a simple and area efficient way of implementing SB multiplication over large Galois Field. It is a LFSR based multiplier. The nice bit-slice depicted in Figure. 1, simplifies the VLSI design for large field arithmetic. The input elements $A(x)$ and $B(x)$ and the output product $C(x)$ are bit serial and the computation proceeds in bit-parallel fashion by convolution and reduction modulo an irreducible polynomial $p(x)$ of degree m. For more details about the algorithm, see [4].

The multiplication is performed with order $O(m)$ in both computation time and implementation area. $2m$ time units are required between the first-in and first-out of computation and two-bits control signal is required. The MSR architecture is programmable with respect to the primitive polynomial $p(x)$ and field order m using extra gates in each multiplier cell [4]. The complexity can be further reduced for implementations that use irreducible polynomials with few coefficients such as *trinomials* or *pentanomials*.

Figure 1

Fig. 1 MSR m-bit multiplier architecture.

In addition, it is easy to make the multiplier work as squarer since squarer can be realized as a bit-serial multiplier [4]. This, simplify the design of the MSR based exponentiator in which, squaring can be carried out concurrently with the multiplication.

Another architecture LSA performing SD multiplication in $GF(2^m)$ is described in [11]; this architecture is linear systolic array, bit-level pipelined Fig.2. It is also highly regular and expandable and performs the SD multiplication operation over $GF(2^m)$ in bit serial manner using the recursive algorithm in (4). It allows the input elements to enter a linear systolic array in the same order and the system only requires one bit pipelined control signal. For more details about the algorithm, see [3].

$$c_i^{(k)} = \begin{cases} c_{m-1}^{(k-1)} p_i + a_i b_{m-1-k} + c_{i-1}^{(k-1)}, & 0 < i \le m-1; \\ c_{m-1}^{(k-1)} p_0 + a_0 b_{m-1-k}, & i = 0 \end{cases} \quad (4)$$

$$c_i^{(-1)} = 0 \ \ for \ 0 \le i \le m-1$$

Where c_i^{m-1} for $0 \le i \le m-1$ (output of the last cell-k) are the coefficients of the product $C(x) = A(x) \cdot B(x) \bmod p(x)$ and p_i for $0 \le i \le m-1$ are the coefficients of the polynomial generator.

Figure 2

Fig. 2 Linear systolic array multiplier.

The circuit diagram of CELL-k is shown in Fig.3. Two internal registers b and c are used to hold the bit coefficients b_{m-1-k} and $c_{m-1}^{(k-1)}$ along the operation using the s_{in} signal which mark the start of the multiplication. These coefficients are then used to compute the CELL-k output (4) at the next clock cycle when $s_{in} = 0$. Thus, three registers a, p and s are used to give one time unit delay to the input bit coefficients a_i, p_i and s_i at each CELL-k. The outputs are then triggered using a next register output stage in master slave manner as shown in Fig. 3.

The m-bit multiplication time takes $3m$-1 clock cycles. At $2m$ clock cycles after a_{m-1} and b_{m-1} enter the leftmost cell; the results will start coming out from the rightmost cell at the rate of one coefficient every clock cycle.

Figure 3

Fig. 3 LSA basic processing cell and its algorithm.

This multiplier is programmable with respect to the primitive polynomial $p(x)$. The algorithm is often advantageous because of its efficient implementation time. The critical path in this architecture is just the sum of one full-adder and one NAND gate delay.

In contrast with the MSR architecture, LSA multiplier can achieve consecutive overlapped multiplication operations without waiting for the result to start a new

computation. Hence, with a simple multiplier and minor modification we can implement the exponentiation based on repeated square-and-multiply algorithm yielding to decrease the exponentiator area when large field-size is used.

3. Architecture Level Transformation

Power consumption in standard CMOS technology originates from two different sources:

- Static power is dissipated in several ways. The largest percentage of static power results from source-to-drain sub threshold leakage, which is caused by reduced threshold voltages that prevent the gate from completely turning off. Static power is also dissipated when current leaks between the diffusion layers and the substrate. For this reason, static power is often called *leakage power*.
- Dynamic power caused by charging and discharging capacitors during signal computation (*Switching power*). *Short circuits* (internal power) occurs also in the dynamic phase where both the nMOS and pMOS transistors are conducting, and

Hence, the total power dissipated in a CMOS gate with a capacitive load C_{load} is given by

$$P = \frac{1}{2} \cdot C_{load} \cdot V_{DD}^2 f \cdot N + Q_{sc} \cdot V_{DD} \cdot f \cdot N + I_{leak} \cdot V_{DD} \qquad (5)$$

Where V_{DD} denotes the voltage swing, and f is the frequency of operation, N the activity factor, i.e., the number of gate output transitions per clock cycle. The factor Q_{sc} represents the quantity of charge carried by the *short circuit* current per transition and I_{leak} is the leakage current.

In traditional design the average power consumption of a CMOS gate is dominated by the switching activity (dynamic power) and contributes to more than 90% of the total power consumption [17]. For recent technologies (deep sub-micron) *short circuit* current and *leakage current* may be neglected, but this may change for future developments of high scaled integration [20]. As the device size and threshold voltage continue to decrease, the short circuit power dissipation is no longer a negligible factor. Reducing the power consumption amounts to the

reduction of one or more of these factors. In energy-efficient design, we seek to minimize the energy consumed per operation or the power-delay product of the circuit, which is the factor of merit for high performance architectures.

Lower supply voltages can achieve extremely low power consumption (5). However, lowering supply voltage leads to performance degradation. Delays drastically increase as V_{DD} approaches the threshold voltages V_t of the device (6).

Since the delay time is proportional to $1/V_{DD}$, the supply voltage can be reduced to a certain value, so that the chosen frequency matches with the longest critical path. The propagation delay equation of a CMOS circuit is given by [21],

$$T_{delay} = \frac{C_{load} \cdot V_{DD}}{k \cdot (V_{DD} - V_t)^2} \qquad (6)$$

Where k depends on the transistors aspect ratio (W/L) and other device parameters, V_t is the transistor threshold voltage.

When the propagation delay is less than the clock period by a factor d, we can reduce the supply voltage by a factor b such that T_{clk} is equal to T_{delay}. Hence,

$$T_{clk} = d.T_{delay}(V_{DD}) = T_{delay}(b \cdot V_{DD}) = \frac{C_{load} \cdot b \cdot V_{DD}}{k \cdot (b \cdot V_{DD} - V_t)^2} \qquad (7)$$

Parallelism and pipelining can be exploited to improve the performance (to compensate for the increased gate delays) of low-voltage circuits [17] [18]. Also, much higher reductions in power consumption are possible when using clock-gating technique in order to reduce the activity factor N in (4). Further, increasing the concurrency of internal operations, and rearranging the gate topology from array-type to tree-type reduces the switching power [15].

In this section, we demonstrate that highest gain can be achieved on the behavioural and architectural levels (up to 90% of power saving) using digit-serial technique to implement partially parallel architecture. We extend the MSR and LSA bit serial multipliers to a generalized digit-serial architecture, which is array-type at the digit-level using parallel multiplication algorithm inside of each digit cells. These architectures are obtained by unfolding the bit-serial multipliers. Instead of the LSA multiplier, the MSR digit-serial architecture cannot be pipelined below digit-level because of the presence of the feedback loops in the

MSR bit-serial architecture. The linear dependency in $mod\ p(x)$ degree reduction operation can be broken by using the trinomials as field-generating polynomials.

3.1. Digit-Serial MSR Multiplier

The architecture presented in Fig. 1 is not pipelined below bit-level. The presence of long loaded lines for large m affects directly the maximum clock frequency and consequently the system performance. In order to overcome this disability, a digit-serial technique can be applied by unfolding the bit-serial MSR architecture.

The transformation approach involves treating the multiplier operands as digits: the m bits of data operands are processed in units (digits) of digit size D using $d = \lceil m / D \rceil$ slices. Let

$$A = \sum_{i=0}^{m-1} a_i x^i, B = \sum_{i=0}^{d-1} B_i x^{Di}, \text{ where}$$

$$B_i = \begin{cases} \sum_{j=0}^{D-1} b_{Di+j} x^i, & 0 \le j \le d-2 \\ \sum_{j=0}^{m-1-D(d-1)} b_{Di+j} x^i, & j = d-1 \end{cases} \tag{7}$$

Then

$$C = A \cdot B \bmod p(x) = A \cdot \sum_{i=0}^{d-1} B_i x^{Di} \bmod p(x) \tag{8}$$

This result on array-type multiplication, which can be performed in the following way:

$$\begin{aligned} C = &\left[B_0 A(x) \bmod p(x)\right] + \left[B_1 (A(x) \cdot x^D \bmod p(x)\right] \\ &+ \left[B_2 \cdot (\cdot x^D (A(x) \cdot x^D \bmod p(x)))\right] + \cdots \\ &+ \left[B_{d-1} (x^{D(d-2)} (A(x) \cdot x^D \bmod p(x)))\right] \end{aligned} \tag{9}$$

We define now the polynomials $Z_{-,j}(x)$ as:

$$Z_{-,j}(x) = \sum_{i=0}^{d-1} z_{i,j}(x^D)^i = (x^D)^j A(x) \mod p(x), j = 0,1,..., m-1 \qquad (10)$$

where $z_{i,j} \in GF(2)$. Then

$$C(x) = \sum_{j=0}^{d-1} B_j Z_{-,j}(x) \qquad (11)$$

And in matrix notation

$$C = \begin{pmatrix} C_0 \\ C_1 \\ . \\ . \\ C_{d-1} \end{pmatrix} = \begin{pmatrix} z_{0,0} & z_{0,1} & . & . & . & z_{0,d-1} \\ z_{1,0} & z_{1,1} & . & . & . & z_{1,d-1} \\ . & . & . & . & . & . \\ . & . & . & . & . & \\ z_{d-1,0} & z_{d-1,1} & . & . & . & z_{d-1,d-1} \end{pmatrix} \bullet B = Z \bullet B \qquad (12)$$

Where Z is a d by d digit matrix. The columns of Z are the d consecutive states of a Galois-type parallel LFSR with feedback polynomial $p(x)$ that has been initially loaded with A (=$Z_{-,0}$). The product is therefore obtained by first loading the LFSR with A, computing $B_0 Z_{-,0}$ and storing the result in d stage register of D-bits size. Next we clock the LFSR, compute $B_1 Z_{-,1}$, add it to $B_0 Z_{-,0}$, and store the result and so forth. After d clock cycles the product is available in the lower register. The general form of the circuit is shown in Fig. 4., for large "Mersenne" prime using trinomial primitive polynomial, with appropriate choice of parameter k (m, k are selected from Table 1.). The structure is kept simple and highly regular.

The explanatory notes for the italic line / across the signal lines denote the weights of the corresponding signals, i.e., $< D - X_1 >_{LSB}$ means that the corresponding line carries the D-X_1 least significant bits of the corresponding signal. The values of X_1 and X_2 are reported in Table 2, with respect to the value of parameter k, the field-size m and digit-size D.

Figure 4

Fig. 4 Digit-serial MSR multiplier for field-generating polynomial p(x)= $1+x^k+x^m$.

The LFSR performs the computation (11), i.e., *A(x)* multiplied by x^D followed by *mod* $p(x)$. The partial product generator denoted by $\otimes$ computes $B_i Z_{-,i}$ in (12). The accumulator is denoted by $\oplus$ and performs the sum operation in (12); it consists

of XOR gates rearranged from array-type to tree-type and storage elements, where the partial product $B_iZ_{-,i}$ and the intermediate result are accumulated using the binary-tree of XOR gates. At each cell, only the D LSB-bits of the partial product are computed. At the last cell, a correction must be done in order to reduce the degree of the result from $m+D-2$ to $m-1$. This can be done efficiently in one step. The polynomial degree is reduced using AND and XOR gates (1). The total computation time takes $3d$ clock cycles between the first-in digit and the last-out digit.

Table 2 x_1 and x_2 values for digit size D = 8,16,32

Table 2

3.2. Linear Digit-Serial Systolic Array Multiplier

The architecture shown in Fig. 4, is not programmable on $p(x)$, *which* is hardwired. We propose here a methodology to design a programmable digit-serial Finite Field multiplier with respect to the primitive polynomial. The multiplier is based on the architecture shown in Fig.2. The digit-serial architecture is obtained by folding the bit-serial architecture implementing (6).

Consider the structure of the bit-serial multiplier shown in Fig. 2. The transformation approach involves treating the bits in this multiplier as digits. Therefore, the inputs bit a_i , b_i , c_i and p_i for $0 \le i \le m-1$ to CELL-k in Fig. 3, are replaced by digits forms A_i, B_i, C_i, P_i for $0 \le i \le d-1$ where

$$< A, B, C, P >_i = \begin{cases} \sum_{j=0}^{D-1} < a, b, c, p >_{Di+j} \; x^j, \; 0 \le i \le d-2 \\ \sum_{j=0}^{m-1-D(d-1)} < a, b, c, p >_{Di+j} \; x^j, \; i = d-1 \end{cases} \tag{13}$$

and

$$\begin{aligned} A(x) &= \sum_{i=0}^{m-1} a_i x^i = \sum_{i=0}^{d-1} A_i x^{Di}, \; B(x) = \sum_{i=0}^{m-1} b_i x^i = \sum_{i=0}^{d-1} B_i x^{Di} \\ C(x) &= \sum_{i=0}^{m-1} c_i x^i = \sum_{i=0}^{d-1} C_i x^{Di}, \; P(x) = \sum_{i=0}^{m-1} p_i x^i = \sum_{i=0}^{d-1} P_i x^{Di} \end{aligned} \tag{14}$$

where, D denotes the digit-size and d the total number of digits, $d = \lceil m / d \rceil$.

Suppose that the resulting architecture can be implemented on a linear digit-serial systolic array, as shown in Fig.5. The inputs digit-words A_i, B_i, C_i, P_i are fed into the multiplier in the same order for i deceasing and from the MSB to the LSB. If m is not divisible per D, the zero padding is performed at the LSB positions for $i = 0$.

Figure 5

Fig. 5 Digit-serial, linear systolic array multiplier.

The system now consists of d identical cells for $D \cdot d$-bit multiplication in $GF(2^m)$. It inputs the data at the leftmost cell and outputs the results at the rightmost cell at the rate of one digit every clock cycle.

The basic processing element CELL-K of the multiplier is shown in Fig. 6. Two D-bit registers A, P and 1-bit s registers are used to give one time unit delay to the input data A_i, P_i and s_i at each CELL-k. The s signal is used to denote the start of a multiplication.

Figure 6

Fig. 6 LSA digit-serial basic processing cell and its algorithm.

The corresponding algorithm is obtained by grouping each set of D cells from the LSA multiplier in Fig. 2, then computing the outputs of each of these grouped cells after D steps (clock cycles). These are the outputs of the resulting digit cell. This is illustrated in the example bellow.

Example 1.

Consider the computation of $C(x) = A(x).B(x) \mod p(x)$ over $GF(2^7)$ where $A(x) = 1 + x^2 + x^4 + x^6$ and $B(x) = x^1 + x^3 + x^5$ and $p(x) = 1 + x^3 + x^7$.

Consider now the basic processing element CELL-k and its algorithm given in Fig. 3. Let each CELL-k be represented using the I/O signals and the state of its internal registers as shown in Fig. 6.

Figure 7

Fig. 7 LSA CELL-K multiplier representation.

The I/O signals and the state of internal registers at each CELL-k for each computation step are reported in Fig. 7. The steps represented at the right side of each CELL6 represents the steps corresponding to the digit-serial architecture for D=3.

Therefore, by folding the bit-serial computations in Fig. 7., and after 3d-1 steps the output of the multiplier expressed in digit form is as follows:

$$c_{out} \rightarrow \textit{000 000 000 000 000 } \mathbf{011\ 001\ 0}\textit{00}$$

Thus, the C internal register of each CELL-k in the digit-serial multiplier are expressed as follows:

LSA multiplier cell LSA multiplier time step

$$C(0) = c_{in}(0,0) \rightarrow C(0) = C_{in}(2)$$

$$C(1) = c_{in}(1,2) = c_{out}(0,1) = c_{in}(0,1) \cdot \bar{s}_{in}(0,1) + c(0,0) \cdot p(0,0) + b(0,0) \cdot a(0,0)$$

$$= c_{in}(0,1) \cdot \bar{s}_{in}(0,1) + c(0) \cdot p(0,0) + b(0) \cdot a(0,0)$$

$$\rightarrow C(1) = C_{in}(1) + (C_{in}(2) \cdot P_{in}(2)) + (B_{in}(2) \cdot A_{in}(2))$$

$$C(2) = c_{in}(2,4) = c_{out}(1,3) = c_{in}(1,3) \cdot \bar{s}_{in}(1,3) + c(1,2) \cdot p(1,2) + b(1,2) \cdot a(1,2)$$

$$= c_{in}(1,3) \cdot \bar{s}_{in}(1,3) + c(1) \cdot p(1,2) + b(1) \cdot a(1,2)$$

where, $c_{in}(1,3) = c_{out}(0,2) = c_{in}(0,2) \cdot s_{in}(0,2) + c(0) \cdot p(0,1) + b(0) \cdot a(0,1)$

$$\rightarrow C(2) = (C_{in}(0) + (C_{in}(2) \cdot P_{in}(1)) + (B_{in}(2) \cdot A_{in}(2))) + [(C_{in}(1) + (C_{in}(2) \cdot P_{in}(2)) + (B_{in}(2) \cdot A_{in}(2))) \cdot P_{in}(2)] + (B_{in}(1) \cdot A_{in}(2))$$

Fig. 8 Folding bit-serial computations.

Note that all the states of C registers must be computed during one clock cycle. The C_{out} register at the output of each CELL-k, are then expressed as follows :

$$C_{out}(2) = ((((C_{in}(2)+(C(0) \,.\, P(0))+(B(0) \,.\, A(0)))+(C(1) \,.\, P(1)) + (B(1) \,.\, A(1))) \,.\, \overline{s}_{in}\,)+(C(2) \,.\, P(2))+(B(2) \,.\, A(2)));$$

$$C_{out}(1) = ((((C_{in}(1)+(C(0) \,.\, P_{in}(2))+(B(0) \,.\, A_{in}(2))) +(C(1) \,.\, P(0)) +(B(1) \,.\, A(0))) \,.\, \overline{s}_{in}\,)+(C(2) \,.\, P(1)) +(B(2) \,.\, A(1)));$$

$$C_{out}(0) = ((((C_{in}(0)+(C(0) \,.\, P_{in}(1))+(B(0) \,.\, A_{in}(1))) +(C(1) \,.\, P_{in}(2)) +(B(1) \,.\, A_{in}(2))) \,.\, \overline{s}_{in}\,)+(C(2) \,.\, P(0)) + (B(2) \,.\, A(0)));$$

Hence, we can extend these expressions to a D-bits digit words and drive a generalized algorithm described bellow, by computing the expression of F and G functions reported in Fig. 9 and 10.

Fig. 9 Circuit diagram of the F function.

In Fig. 5, F denotes the function processing the state of the C internal register. The circuit diagram of F function is shown in Fig. 9., where FF denotes a flip-flop.
Note that the critical path is $(D\text{-}1)(T_{xor\text{-}3}+T_{NAND\text{-}2})$.

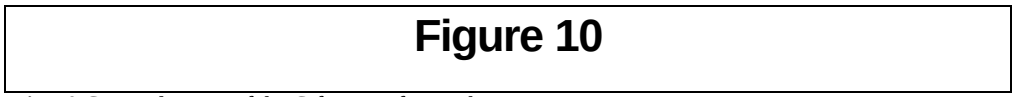

Fig. 10 Circuit diagram of the G function for one bit output.

The G function process the state of the output register C_{out}. The corresponding circuit diagram for one output coefficient is shown in Fig. 10. The critical path in this architecture is increased to $D\,T_{XOR\text{-}3}+T_{XOR\text{-}2}+2T_{NAND\text{-}2}$

The d-bit multiplication implemented within architecture shown in Fig. 5., takes $3d$-1 clock cycles. At $2d$ clock cycles after $A_{d\text{-}1}$ and $B_{d\text{-}1}$ enter the leftmost cell, the

results will start coming out from the rightmost cell at the rate of one digit every clock cycle.

4. Implementation Issues and Comparison

Clock gating technique can be used for power-efficient implementation of registers that are disabled during some clock cycles, when such registers maintain the same value through multiple cycles such as the internal slave registers c and b in Fig. 3 and C and B in the LSA architecture shown in Fig. 6. These registers have their own load controlled by s_{in} signal. This technique works well for data-flow logic, where clocking requirements can be predetermined at least one cycle ahead. Thus, the clock gating enable signal s_{in} must be valid halfway into the cycle to gate off the capture clock. To overcome this problem, we require that these internal registers be triggered faster than the master registers B_{out} and C_{out} using different clock edges that is pipelining within the clock cycle. This requires one more clock pulse, resulting in 2-phase non-overlapping clocking scheme.

The MSR, LSA and clock gated LSA architectures have been implemented at the gate level using different digit-size $D=1,4,8,16$ in order to perform a comparison in terms of speed, area and energy consumption for $GF(2^{607})$ multiplier with $p(x)=1+x^{273}+x^{607}$ as primitive polynomial. We mapped our design into a deep sub-micron (0.18µ) target library from XEMICS (COOLIB) that contains rich logic-gates optimised for low-power/low-voltage, operating at two different power supply 1.8v and 0.9v. A low power design-flow has been validated using Synopsys tools for power analysis and optimization.

Different types of power dissipation components are estimated using gate level simulations on a set of random stimulus. Since *low-energy* design is more important than *low-power* design, the energy and energy-delay product is computed. The performance characteristics including total delay, area in term of gates and energy-delay product and are reported in Fig. 11, 12 and 13 respectively.

Figure 11

Fig. 11 Total delay comparison as function of the digit-size for one $GF(2^{607})$ multiplication.

Note that, the long signals that are distributed to all slices in Fig. 1 are susceptible to degradation due to the large capacitive loads. For example, the serial input multiplier bit b_i is a long line that has to drive m AND gate. This signal must drive up to 11.76pF capacitive load. The source of this line will be trying to push current into the entire load and experiencing a very substantial RC delay, which, increase considerably the critical path and then the total delay as shown in Fig. 11. Hence, for large m, a number of refresh amplifiers is clearly needed to manage such a heavy load. We can consider using fast buffers to isolate heavy loads. However buffering the architecture can severely degrade system performance, it increases the critical path and creates the problem of skewed signals. Thus, when using a buffer, trace lengths should be balanced to minimize signal skew. The parallelism inside each digit-cell in Fig. 4 contribute to reduce the load on such long heavily loaded signals, i.e., when the chosen digit-size is 8 the capacitive load is significantly reduced to 1,35pF per bit-line for the most heavily loaded line (input multiplier digit-word B_i), experiencing over 72% improvement in circuit speed when operating at 1.8v and 96% when operating at 0.9v.

Figure 12

Fig. 12 Area in gates of the MSR, LSA and clock gating LSA digit-serial GF(2^{607}) *multipliers* as function of the digit size.

The bit-level pipelining approach makes the LSA multiplier architecture more advantageous in term of clock frequency and computation time. Both the total delay and the energy consumption are reduced. On the one hand, the latency decrease linearly with the digit-size but the critical path increases linearly in almost the same rate (Fig. 11.), resulting in a constant total delay for digit-size equal or larger then 4. On the other hand, the area increases dramatically due to the large number of latches used to temporary hold the internal and the output data in master-slave manner (Fig. 13). This means that the level of parallelism is limited by the area constraints.

Figure 13

Fig. 13 Energy-Delay product comparison between MSR and LSA digit-serial GF(2607) multipliers.

The most interesting result is obtained when comparing the Energy-Delay and the Energy-Delay-Area products. The performance characteristic reported in Fig. 13, shows that Energy-Delay products are significantly reduced for both LSA and MSR architecture when digit-size increase. High gain is obtained for D=16 when operating at 0.9v and more than 99% reduction is noticed. However, when comparing the characteristic reported in Fig. 14, the optimum gain for LSA architecture is obtained for D=4 when operating at 1.8v due to the dramatic increase in circuit area for larger digit-size. This is not the case when operating at 0.9v since the energy is significantly reduced.

Beside the programmability with respect to the primitive polynomial of the LSA architecture, the MSR present the best performance characteristic only for digit-size equal or larger then 8 due to the large critical path for small digit-size.

The clock gating technique inserted for LSA multiplier achieves a substantial reduction in both the Energy-Delay (over 28% at 0.9v and 17% at 1.8v for D=8) and the Energy-Delay-Area product (over 30% at 0.9v and 20% at 1.8v for D=8) for only 22% of clock gated registers. Clock gating reduces the number of gates in such architecture (multi-bit registers) when digit-size increase. It helps to eliminate the feedback loops and multiplixers used to feedback the output of each internal storage elements back to the input for synchronous load-enable registers. Such feedback loops and multiplixers are replaced by only one integrated cell with latch based clock gating which result in 3.5% and 4.3% reduction in gate number at 0.9v and 1.8v respectively when the chosen digit-size is 8.

Figure 14

Fig. 14 Energy-Delay-Area product comparison between MSR and LSA digit-serial GF(2^{607}) multipliers.

When reducing the operating voltage by a factor δ=2 the switching power is reduced by factor δ^2 (5), from (6) assuming that $V_{DD} >> V_t$ the delay is increased by factor δ. If the switching power contribute to more then 90% (dominant factor)

then, the power saving is counterbalanced by the increased delay since the energy-delay product is proportional to δ^2.

5. Conclusion

The VLSI architectures presented here are low energy, digit-serial, suitable for large prime $GF(2^m)$ multiplication. The MSR architecture is area efficient LFSR-based for trinomial polynomial field-generator and the LSA architecture is bit-level pipelined, linear systolic array architecture, which is programmable with respect to the primitive polynomial $p(x)$.

Digit-serial technique when applied to the MSR architecture can be exploited efficiently in order to decrease the critical path (total delay), when buffering the architecture, and helps to reduce the switching activity for the LSA architecture. This results in low energy design of large finite-field multipliers at the expense of increased area. Higher gain in energy-delay product is obtained (over 90%) when digit-size is large. Therefore, a trade-off can be made between the area, energy consumption and speed. No significant gain on the energy-delay product is obtained when reducing voltage supply since the delay and power counterbalance each other when the switching power dominates. Gating the clock when possible achieves a great saving in power consumption and area and has no significant effect on the circuit speed.

References

[1] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press LLC, 1997.

[2] B. Schneier, *Applied Cryptography*, Second Edition, Wiley, 1996.

[3] A. M. Odlyzko, *Discrete Logarithms in Finite Fields and their cryptographic significance*, EUROCRYPT, 1984, pp. 224-314.

[4] E. D. Mastrovito, *VLSI Architectures for Computations in Galois Fields*, PhD thesis, Linköping University, Departement of Electrical Engineering, Linköping, Sweden, 1991.

[5] S. K. Jain, L. Song, K. K. Parhi, *Optimum Primitive Polynomials for Low-Area Low-Power Finite Field Semi-Systolic Multipliers*, ECE Department, University of Minnesota, Minneapolis.

[6] M. Kovac, N. Ranganathan, *ACE: A VLSI Chip for Galois Field Efficient GF(2^n) Based Exponentiation*, IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 43, No. 4, pp. 189-297, April 1996.

[7] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Volume2, Addison-Wesley, Second edition, 1981.

[8] Sebastian T. J. Fenn, Mohammed Benaissa, and David Tylor, *GF(2^m) Multiplication and Division*, IEEE Transactions on Computers, vol. 45, no. 3, pp. 319-327, March 1996.

[9] Chin-Liang Wang. *Bit-Level Systolic Array for Fast Exponentiation in GF(2^m)*, IEEE Transactions on Computers, vol. 43, no. 7, pp. 838-841, July 1994.

[10] S. K. Jain, L. Song, K. K. Parhi, *Efficient Semi-Systolic Architectures for Finite Field Arithmetic*, IEEE Trans. on VLSI Systems, vol. 6, pp. 101-113, March 1998.

[11] [11] B. B. Zhou, *A New Bit-Serial Systolic Multiplier Over GF(2^m)*, IEEE Transactions on Computers, Vol. 37, No. 6, pp. 749-751, June 1988.

[12] I. S. Hsu, T. K. Truong, L. J. Deutsch, and I. S. Reed, *A Comparison of VLSI Architecture of Finite Field Multipliers Using Dual, Normal, or standard Bases*, IEEE Transactions on Computers, vol. 37, no. 6, pp. 735-739, June 1988.

[13] Christof Paar, Nikolaus Lange, *A Comparive VLSI Synthesis of Finite Field Multipliers*, Proceedings of the 3rd International Symposium on Communication Theory & Applications, 10-14 July 1995, Lake District, UK.

[14] A. G. Wassal, M. A. Hassan and M. I. Elmasry, *Low-Power Design of finite Field Multipliers for wireless Applications*, Proceedings of the Great Lakes Symposium on VLSI '98.

[15] L. Song, K. K. Parhi, *Low-Energy Digit-Serial/Parallel Finite Field Multipliers*, Journal of VLSI Signal Processing Systems, Vol. 19, N0. 2, Issue No. 2, pp. 149-166, June 1998.

[16] M. R. Schroeder, *Number Theory in Science and Communication*, Volume2, Springer-Verlag, 1986.

[17] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, 1995.

[18] A. P. Chandrakasan and R. W. Brodersen, *Minimizing Power in Digital CMOS Circuits*, Proceedings of the IEEE, Vol. 83, No. 4, April 1995.

[19] A. M. Odlyzko, *Discrete logarithms: The past and the future*, Designs, Codes and Cryptography 19(2/3), 2000, pp. 129-147.

[20] Y. Taur, Y. J. Mii, D.J. Frank, H.S. Wong, D.A. Buchanan, S.J. Wind, S. A. Rishton, G.A. Sai-Halasz and E.J. Nowak, *CMOS scaling into the 21st century: 0.1 mm and beyond*, IBM Journal of Research and Development, vol. 39, no. ½, Jan/Mar 1995, pp. 245-260.

[21] A. Bellaouar, M. Elmasry, *Low-Power Digital VLSI design: Circuits and Systems*, Boston, Massachusetts, Kluwer Academic Publishers.

Figure Captions

Fig. 1 MSR *m*-bit multiplier architecture.

Fig. 2 Linear systolic array multiplier architecture.

Fig. 3 LSA basic processing cell and its algorithm.

Fig. 4 Digit-serial Multiplier for field-generating polynomial $p(x) = 1 + x^k + x^m$.

Fig. 5 Digit-serial, linear systolic array multiplier.

Fig. 6 Digit-serial LSA basic processing cell and its algorithm.

Fig. 7 LSA CELL-K multiplier representation.

Fig. 8 Folding bit-serial computations.

Fig. 9 Circuit diagram of the *F* function.

Fig. 10 Circuit diagram of the G function for one bit output.

Fig. 11 Energy-Delay product comparison between MSR and LSA digit-serial GF(2607) multipliers.

Fig.12 Total delay comparison as function of the digit-size for one $GF(2^{607})$ multiplication.

Fig. 13 Area in gates of the MSR, LSA and clock gating LSA digit-serial $GF(2^{607})$ *multipliers* as function of the digit size.

Table Captions

Table 1 Most useful irreducible trinomials $x^m + x^k + 1$, for each large Mersenne prime m, $512 \leq m \leq 4423$

Table 2

Figure 1

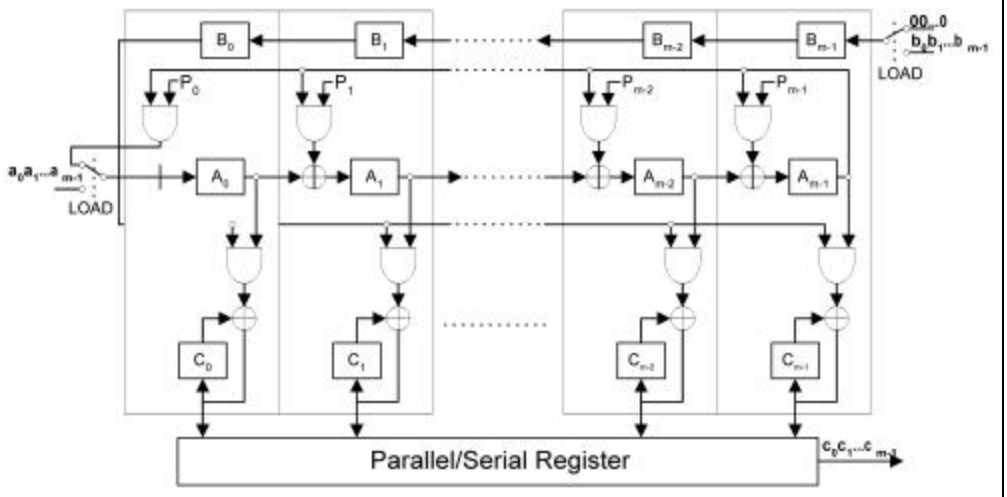
B0
B1
Bm-2
Bm-1
00..0
b0b1..bm-1
LOAD
P0
P1
Pm-2
Pm-1
a0a1..am-1
LOAD
A0
A1
Am-2
Am-1
C0
C1
Cm-2
Cm-1
Parallel/Serial Register
c0c1..cm-1

Figure 2

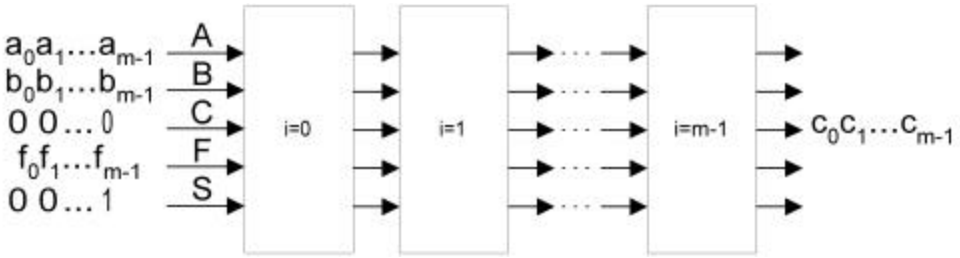
$a_0a_1\ldots a_{m-1}$
$b_0b_1\ldots b_{m-1}$
0 0 ... 0
$f_0f_1\ldots f_{m-1}$
0 0 ... 1
A
B
C
F
S
i=0
i=1
i=m-1
$c_0c_1\ldots c_{m-1}$

Figure 3

if s_{in} then
  begin
  $b := b_{in}$; c := c_{in} ;
end;
$b_{out} := b_{in} \cdot s_{in}$;
$c_{out} := c_{in} \cdot s_{in} + c \cdot p + b \cdot a$;
$a := a_{in}$; $a_{out} := a$;
$p := p_{in}$; $p_{out} := p$;
$s := s_{in}$; $s_{out} := s$;

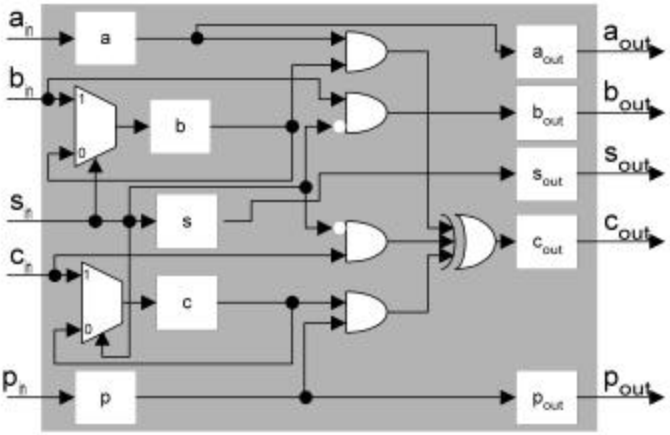

Figure 4

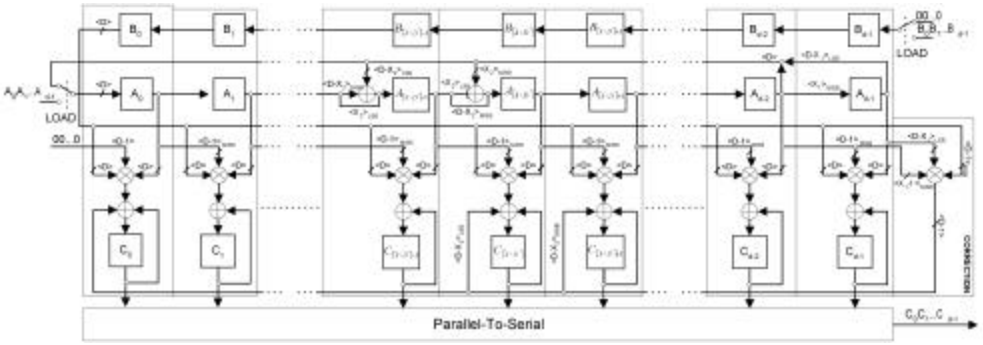

Figure 5

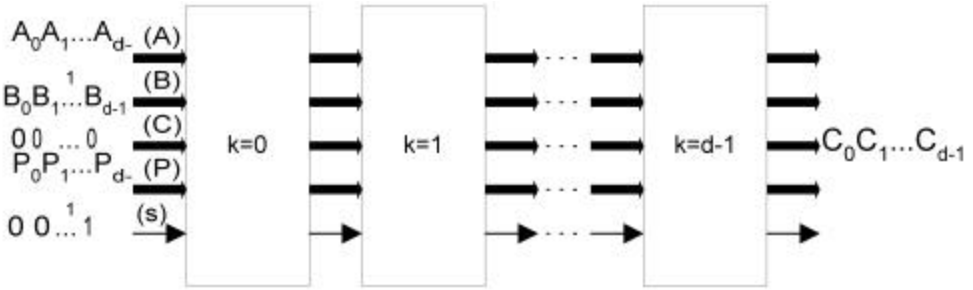

Figure 6

```
if s_in then
  begin
    B(0 to D-1) := B_in(D-1 to 0);
    for i = 0 to D-1
      C(i) := F(C_ie, P_ie, B_in, A_in);
end;
B_out := B_ie;
A := A_in; A_out := A;
P := P_ie; P_out := P;
s := s_ie; s_out := s;
for i = 0 to D-1
  C_out(i) := G(s_in, C_ie, C, P_in, P, A_in, A, B);
```

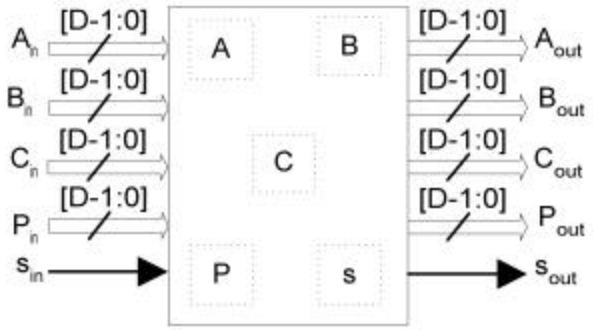

Figure 7

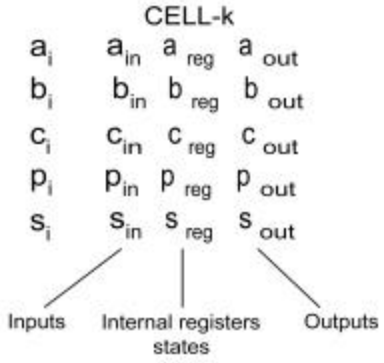
CELL-k
a_i a_{in} a_{reg} a_{out}
b_i b_{in} b_{reg} b_{out}
c_i c_{in} c_{reg} c_{out}
p_i p_{in} p_{reg} p_{out}
s_i s_{in} s_{reg} s_{out}
Inputs
Internal registers states
Outputs

Figure 8

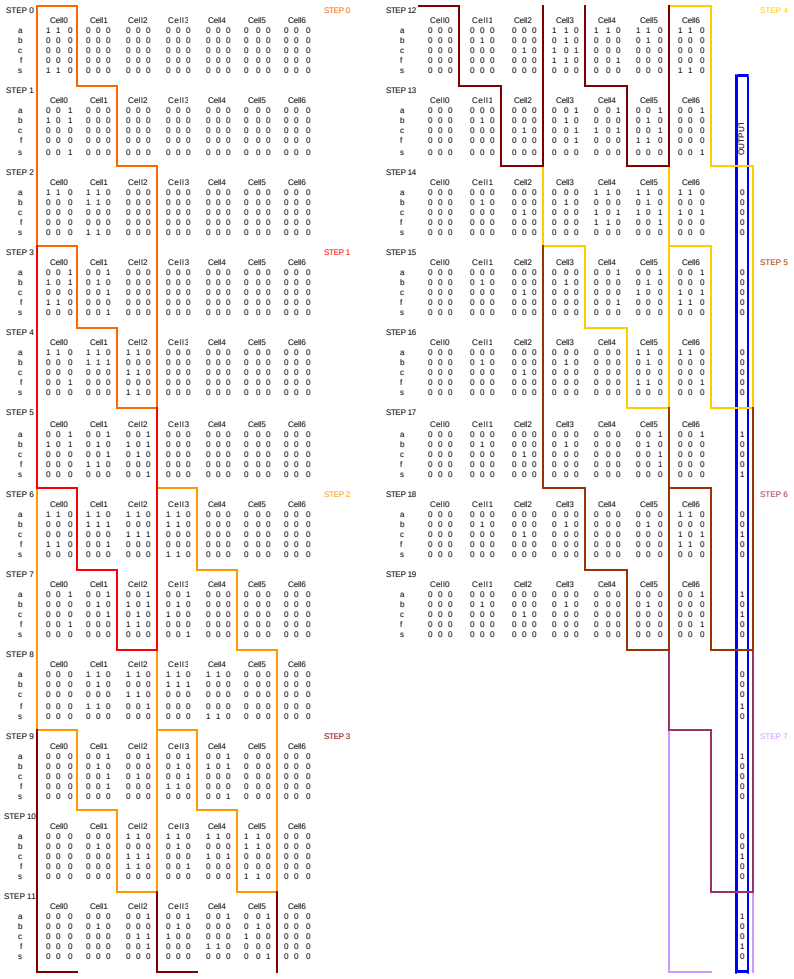

Figure 9

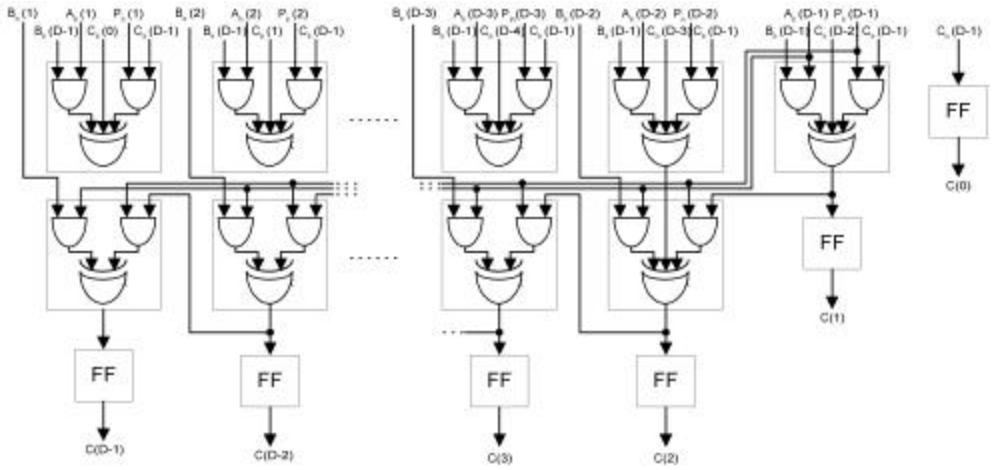

Figure 10

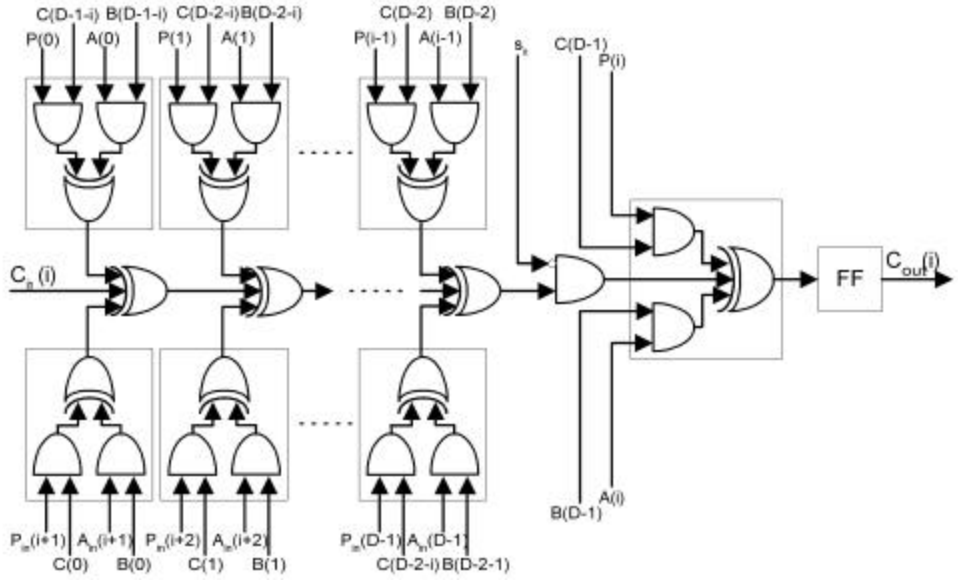
C(D-1-i) B(D-1-i) C(D-2-i) B(D-2-i)
P(0) A(0) P(1) A(1)
C(D-2) B(D-2)
P(i-1) A(i-1)
C(D-1)
P(i)
FF
B(D-1)
A(i)
C(0) B(0) C(1) B(1)
C(D-2-i) B(D-2-1)

Figure 11

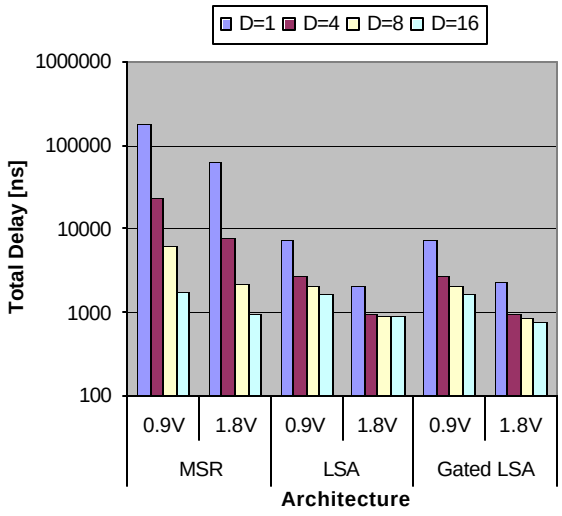
D=1
D=4
D=8
D=16
1000000
100000
10000
1000
100
Total Delay [ns]
0.9V
1.8V
0.9V
1.8V
0.9V
1.8V
MSR
LSA
Gated LSA
Architecture

Figure 12

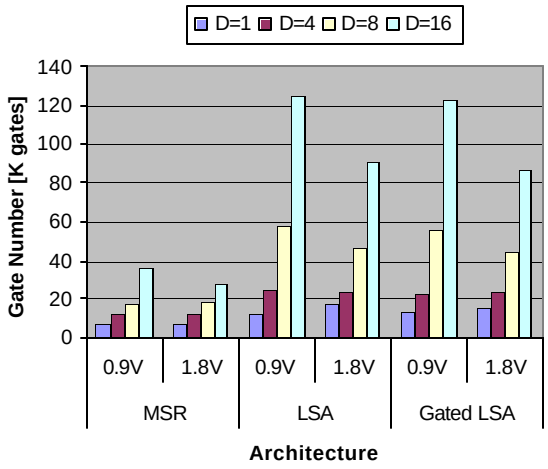
D=1
D=4
D=8
D=16
Gate Number [K gates]
140
120
100
80
60
40
20
0
0.9V
1.8V
0.9V
1.8V
0.9V
1.8V
MSR
LSA
Gated LSA
Architecture

Figure 13

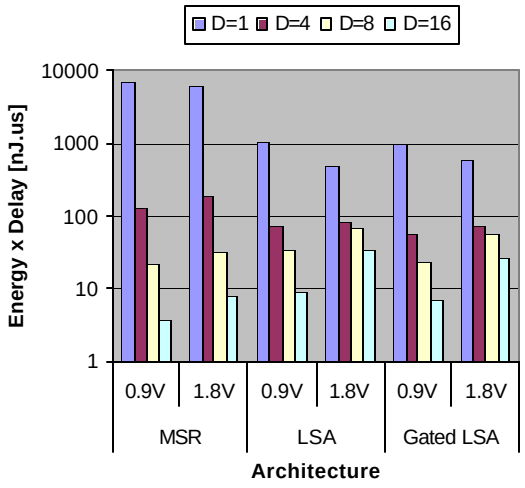
D=1
D=4
D=8
D=16
10000
1000
100
10
1
Energy x Delay [nJ.us]
0.9V
1.8V
0.9V
1.8V
0.9V
1.8V
MSR
LSA
Gated LSA
Architecture

Figure 14

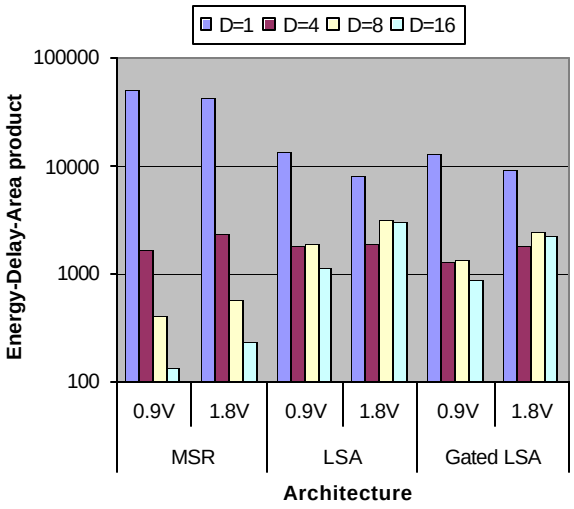
D=1
D=4
D=8
D=16
100000
10000
1000
100
Energy-Delay-Area product
0.9V
1.8V
0.9V
1.8V
0.9V
1.8V
MSR
LSA
Gated LSA
Architecture

Table 1

m	k
521	32, 48, 158, 168, 353, 363, 473, 489
607	105, 147, 273, 334, 460, 502
1279	216, 418, 861, 1063
2281	715, 915, 1029, 1252, 1366, 1566
3217	67, 576, 2641, 3150
4423	271, 369, 370, 649, 1393, 1419, 2098, 2325, 3004, 3030, 3774, 4053, 4054, 4152

Table 2

m	X_1			k	X_2		
	D=4	D=8	D=16		D=4	D=8	D=16
521	7	7	23	32	0	0	0
				48	0	0	16
				158	6	12	30
				168	0	8	8
607	1	1	1	105	1	9	9
				147	3	3	19
				273	1	1	17
1279	1	1	1	216	0	8	24
				418	2	2	2
1281	7	15	31	715	3	11	11
				915	3	3	19
				1029	5	5	5
3281	7	15	15	67	3	3	3
				576	0	0	0
4423	1	9	25	271	7	15	15